C64

Advanced Book of FACTS

THE ADVANCED BOOK OF FACTS

By

GARY STACEY

© Copyright 1985 by G.L. Stacey
ALL RIGHTS RESERVED

Published by: *MegaSoft, Ltd.*
P.O. Box 1080, Battle Ground, WA 98604

The information contained within the pages of this book should prove to be a very useful resource. The five memory monitors I have chosen to explain are among the most popular. Also included are DISK DOCTOR instructions, this is by far the one of the best known sector editors. It is my hope this book will make programing the Commodore a little less mysterious.

INDEX

INDEX

1. Supermon Page 2

2. Extramon Page 6

3. Micromon Page 9

4. Commodore Page 12

5. Hesmon Page 14

6. Disk Doctor Page 18

7. Conversion Chart Page 20

8. Color Memory Chart Page 25

9. Screen Memory Chart Page 26

10. Technical Terminology Page 27

11. Forms of Protection Page 28

12. Diskette Format Page 39

13. Menu Program Page 41

The word COMMODORE is a registered trade mark of COMMODORE BUSINESS MACHINES. HESMON is a registered trade mark of HUMAN ENGINEERED SOFTWARE.

A memory monitor is a program that enables you to examine and modify the contents of the computers memory. One of the purposes of this is to allow the user to alter the operation of machine language programs. There are many memory monitor programs that can be used, some are in the public domain, and others are sold by commercial software companys. Not all monitors are alike, the functions and the number of functions vary. Monitors are loaded in to memory from either tape, disk, or cartridge. Below you will find a list of the monitor commands for each monitor that is covered in this book.

SUPERMON 2/4

Commands

A (Simple Assembler)

```
Example: .A 4000 LDA #$00
         .A 4002 STA $6000
         .A 4005 <PRESS RETURN>
```

The example above starts the assembly at the hex memory location 4000. The instruction that follows will load the accumulator with zero. In the second line the A instruction and the address is prompted by the assembler, and the user is not required to type it in. Press the return key after the address prompt to end the assembly and exit the assembler.

D (Disassembler)

```
Example: .D 4000

         ., 4000 A9 00       LDA #$00
         ., 4002 8D 00 60    STA $6000
         ., 4005 60          RTS
```

The example starts a disassembly starting at hex 4000. The disassembly of 22 instructions will start at this address. The three bytes after the address can be modified by using the screen editor. Moving the cursor to which ever byte that you want to change and type over it, and press the return key to enter into memory. Change the output to the printer to get a hard copy of the disassembly. This is accomplished by doing the following: PRESS THE X KEY TO EXIT THE MONITOR THEN TYPE OPEN4,4:CMD4. Now to reenter the monitor by typing SYS 2048.

I (Single Step)

Allows a machine language program to execute one step at a time. Use the R command to set the program counter address to the instruction that you wish to start single stepping. The I command will start the execution of each instruction one after the other. There are four controls that are used with this command.

K (To Single Step)

SPACE (TO Fast Step)
STOP (To Return To The Monitor)

F (Fill Memory)

Example: .F 4000 5000 00

This command fills the memory from 4000 hex to 5000 hex with zeros. If all memory except where the monitor resides is filled with zeros before loading a program. It is easier to find the starting and ending addresses of the program.

G [Run]

Example: .G 4000

The code at 4000 hex will be run.

Example: .G

This command will go to the address in the program counter register and begin to run the code.

H (Hunt)

Example: .H 4000 5000 (READ)

This command will hunt from 4000 to 5000 hex for the ASCII string READ. All memory locations in the specified range that contain the ASCII string READ will be displayed. A single quote must preceed the ASCII string requested.

Example: .H 4000 5000 A9 00

A search for the data $A9,$00 will be made and the memory locations that contain this data will be reported.

L (Load from tape or disk)

Example: .L"Test"

Loads the program (Test) from cassette.

.L"Test",08

Loads program (Test) from device 8 (disk drive), the basic pointers will not be changed using this command.

M (Memory Display)

```
Example: .M 4000 5000

               . 4000 A9 01 A0 00 99 00 80 99
               . 4008 00 81 99 00 82 99 00 83
               . 4016 C8 D0 F1 60 00 00 00 00
```

This will display the contents of memory from 4000 hex to 5000 hex. The bytes following each address can be modified by using the screen editor. Move the cursor to where you wish to edit, type in any change, then press the return key to enter into memory.

R (Register Display)

```
Example: ,P

               .;  PC   SR AC XR YR SP
               .; 0800 30 00 00 00 F6
```

This displays the register values. These can be changed with the screen editor and entered using the return key. This instruction can be used to set up the program counter register with a value before using the I instruction to single step.

S (Save)

Example: .S"Test",01,0800,1000

Save the contents of memory from 0800 hex up to but not including 1000 hex to tape, using the name (Test.)

Example: .S"Test",08,0800,1000

Same as above but, saves to device 8 (disk drive).

T (transfer Memory)

Example: .T 8000 A000 2000

This will transfer the memory contents of 8000 hex through A000 hex and store it starting with memory at 2000 hex.

X (Exit Back to Basic)

This command will return to the basic ready mode. Care should be taken so that the stack value is the same as when the monitor was entered. Using the basic statement CLR will cure any stack problems.

SUMMARY OF SUPERMON INSTRUCTIONS

A (SIMPLE ASSEMBLER)

D (DISASSEMBLE CODE)

F (FILL MEMORY)

G (GO RUN)

H (HUNT MEMORY)

I (SINGLE STEP INSTRUCTION)

L (LOAD FROM TAPE OR DISK)

M (DISPLAY MEMORY IN HEX)

P (PRINT DISASSEMBLY)

R (DISPLAY CONTENTS OF REGISTER)

S (SAVE TO TAPE OR DISK)

T (TRANSFER CODE)

X (RETURN TO BASIC)

M (DISPLAY CONTENTS OF MEMORY)

R (DISPLAY REGISTER CONTENTS)

S (SAVE TO TAPE OR DISK)

X (EXIT TO BASIC)

EXTRAMON

A (SIMPLE ASSEMBLER)

Same as supermon.

B (Break set)

Example: .B 4000 OOFF

The example sets a break point at 4000 hex on the FF occurence of the instruction at 4000. This command allows you to set a break point as a particular address. A break point will stop execution of the code. Use the break set with the quick trace command.

C (compare Memory)

Example: .C 8000 A000 2000

This will compare the contents of memory from 8000 hex through A000 hex with the contents beginning at 2000. Any memory location that differs will be listed. Usually this command is used to compare an original program file with the same file that has been edited. The procedure is first to load the original, then use the T command to transfer it to a different location. Next load the edited file, and then use the C command to compare the two.

D (Disassemble Memory)

Same as supermon.

F (Fill Memory)

Same as supermon.

G (Go Run)

Same as supermon.

H (hunt Memory)

Same as supermon.

I (Interrogate Memory)

Example:

```
.I 4000 4024

. 4000 48 49 21 20 54 48 45 52 HI! THER
. 4008 45 20 48 4F 57 20 41 52 E HOW AR
. 4016 45 20 59 4F 55 20 20 20 E YOU
```

This displays the hex and the ASCII of memory from 4000 hex through 4024 hex.

Example: .I 4000

Displays memory from 4000 hex to the end of memory.

NOTE - Not the same as supermon.

L (Load From Tape or Disk)
Same as supermon.

M (Display Memory)
Same as supermon.

N (New Locator)
Example: .N 4100 5100 0100 4200 5200
This command is used with the T command which transfers memory. Suppose you transfer memory from 4100 hex through 5100 hex to a location of 4200 to 5200. Any address in the program now points 0100 too high. The new locator command is used to fix this. In the example above, memory locations 4100 hex through 5100 are scanned. The offset of 0100 is added to the adjusted addresses. The memory from 4200 hex through 5200 hex are the adjusted addresses. A (W) after the adjusted addresses is used if the addresses scanned are words and not opcodes. If the (W) is not used when there is a word table then the new locator command will halt.

Q (Quick Trace)
Example:

```
.Q 1000

.Q
```

The first example begins trace from hex 1000. The second begins trace at the address in the program counter register. This command is used after the break point set to debug machine language programs. The G (go) command is similar, but only quick trace command will recognize the break point set command. Also program execution is much slower than the go command. The execution of the quick trace command can be stopped by pressing the stop and (=) key at the same time. NOTE: Not available on all versions.

R (Display Register)
Same as supermon.

S (Save To Tape or Disk)
Same as supermon.

T (Memory Trace)
Same as supermon.

W (Walk Code)
Example:

```
.W 1000
.W
```

The first example will single step the code starting at 1000 hex. The second example will single step the code starting at the address in the program counter. The walk command will execute each instruction and then display the disassembly of the next instruction. The execution speed of the walk command can be controlled by the following keys.

K (Single Step Speed)
RVS (Slow Step Speed)
SPACE (Fast Step Speed)
STOP (Stops Execution)

X (Exit To Basic)
Same as supermon.

SUMMARY OF EXTRAMON COMMANDS

A	(SIMPLE ASSEMBLER)
B	(BREAK POINT SET)
C	(COMPARE MEMORY)
D	(DISASSEMBLE CODE)
F	(FILL MEMORY)
G	(GO RUN)
H	(HUNT MEMORY)
I	(INTERROGATE MEMORY)
L	(LOAD FROM TAPE OR DISK)
M	(DISPLAY MEMORY IN HEX)
N	(NEW LOCATOR)
Q	(QUICK TRACE)
R	(DISPLAY REGISTER CONTENTS)
S	(SAVE TO TAPE OR DISK)
T	(TRANSFER CODE)
W	(WALK CODE)
X	(EXIT TO BASIC)

MICROMON

A (simple Assembler)
Same as supermon and extramon.

B (break Point Set)
Same as extramon.

C (Compare memory)
Same as extramon.

D (Disassemble Memory)
Same as supermon and extramon except that micromon has the ability to scroll up or down using the cursor keys. As the screen is scrolled each new address is displayed and disassembled. This makes debuging programs much easier.

E. (Exit Micromon)
Combine the killing of micromon and the exit to basic.

F. (Fill Memory)
Same as supermon and micromon.

G (Go Run)
Same as supermon and micromon.

H (Hunt Memory)
Same as supermon and micromon (max. of 32 characters.)

L (Load From Disk)
Same as supermon and extramon except that tape loads cannot be done from within the monitor. To load with tape you must exit micromon then load program from tape. After the program has been loaded into memory type SYS 49152 to restart micromon.

M (Display Memory)
Same as supermon and extramon except micromon has the ability to scroll and screen the same as it does with the disasssembly command. You can also toggle between the memory display and the disassembly by using the (:) and (,) alternately. The interrogate function of Extramon is incorporated into Micromon.

N (New Locator)
Same as extramon.

O (Calculate Branch Offset)
Example: .0 4000 5000 FF
The first address is the starting address. The second address is the target address. The branch instructions are calculated and the offset is displayed.

Q (Quick Trace)
Same as extramon.

R (Display Register Contents)
Same as supermon and extramon.

S (Save To Disk)
Same as supermon and extramon except that cassette saves cannot be accomplished within the monitor. NOTE - See load command.

T (Transfer Memory)
Same as supermon and extramon.

W (Walk Code)
Same as extramon (the stop key stops walking and the J key completes a subroutine then continues to walk.

X (Exit To Basic)

SUMMARY OF MICROMON INSTRUCTIONS

A (SIMPLE ASSEMBLER)

B (BREAK POINT SET)

C (COMPARE MEMORY)

D (DISASSEMBLE CODE)

E (EXIT MICROMON)

F (FILL MEMORY)

G (GO RUN)

H (HUNT MEMORY)

L (LOAD FROM DISK)

M (DISPLAY MEMORY IN HEX)

N (NEW LOCATOR)

O (CALCULATE BRANCH OFFSET)

Q (QUICK TRACE)

R (DISPLAY REGISTER CONTENTS)

S (SAVE TO DISK)

T (TRANSFER CODE)

W (WALK CODE)

x (EXIT TO BASIC)
·Below is a list of special commands.

] (LIST DIRECTORY)

+ (add two hex numbers together)
Example: .+ 1111 2222
Answer: .+ 1111 2222 3333

- (Subraction of two hex numbers)
Example: Same as above.

& (Checksum of two addresses are calculated and displayed)
Example: .& 4000 5000
Answer: .& 4000 5000

HEX CONVERSION:
A hex number and its decimal equivalent is input. The ASCII for the two bytes, and the binary are displayed.
Example: .$474F 18255
Answer: .$474F 18255 GO 0100 0111 0100 1111

DECIMAL CONVERSION:
A decimal number and its hex equivalent is input. The ASCII for the two bytes, and the binary are displayed.
Example: .#18505 4849
Answer: .#18505 4849 HI 0100 1000 0100 1001

BINARY CONVERSION:
A binary number is input, and the hex, decimal, and ASCII values are displayed.
Example: .%0101010101010000
Answer: .%0101010101010000 5550 21840 UP

ASCII CONVERSION:
An ASCII character is input, and the hex, decimal, and binary values are displayed.
Example: ."A
Answer: ."A 41 65 0100 0001

COMMODORE MONITOR

A (Simple Assembler)
Same as supermon, extramon and micromon.

C (Compare Memory)
Same as extramon and micromon.

D (Disassemble Memory)
Same as micromon.

F (Fill Memory)
Same as supermon, extramon, and micromon.

G (Go Run)
Same as supermon, extramon, and micromon.

H (hunt Memory)
Same as supermon, extramon, and micromon.

I (Interrogate Memory)
Same as extramon.

L (Load From Tape Or Disk)
Same as supermon and extramon.

M (Display Memory)
Same as supermon and extramon.

N (New Location)
Same as extramon and micromon.

R (Display Register Contents)
Same as supermon, extramon, and micromon.

S (Save To Tape Or Disk)
Same as supermon and extramon.

T (Transfer Memory)
Same as supermon, extramon and micromon.

X (Exit to Basic)

SUMMARY OF COMMODORE MONITOR COMMANDS

A	(SIMPLE ASSEMBLER)
C	(COMPARE MEMORY)
D	(DISASSEMBLE CODE)
F	(FILL MEMORY)
G	(GO RUN)
H	(HUNT MEMORY)
I	(INTERROGATE MEMORY)
L	(LOAD FROM TAPE OR DISK)
M	(DISPLAY HEX VALUE OF MEMORY)
N	(NEW LOCATOR)
R	(DISPLAY REGISTER CONTENTS)
S	(SAVE TO TAPE OR DISK)
T	(TRANSFER CODE)
X	(EXIT TO BASIC)

The name COMMODORE is a registered trade mark of COMMODORE BUSINESS MACHINES.

HESMON

A (Simple Assembler)
Same as all other monitors listed.

B (Break Point Set)
Same as extramon and micromon.

C (Compare Memory)
Same as extramon, micromon, and Commodore

D (Disassemble Code)
Same as micromon and Commodore monitors

E (External Relinker)
Example: .E 4000 5000 2000 8000 9000
This command is used to convert machine language programs from one 6502 based computer to another. Because the memory configuration is not the same on all 6502 based computers, it is necessary to get a rom map of the computer you wish to emulate. If you have a rom table load it using the load command. When no rom table is available, external relinker will use your answers to its questions to construct one. Suppose you want to construct a table at 2000, start by entering four zeros. Use the fill command to do this (F 2000 2003 00). The first two memory locations are the start and end of the program. The third is where the correspondence table starts. The last two memory locations contain the address range that you will be relinking.

F (Fill Memory)
Same as all other monitors listed.

G (Go Run)
Same as all other monitors listed.

H (Hunt Memory)
Same as all other monitors listed.

I. (Interrogate Memory)
Same as micromon.

L (Load From Tape Or Disk)
Same as supermon, extramon, and Commodore monitors.

M (Display Hex Value Of Memory)
Same as Commodore monitor.

N (New Locator)
Same as extramon, micromon, and Commodore monitors.

O (Output Divert)

Example: .0 15 02"FILENAME"

The output of data can be sent to any device. If an 0 is typed the output will be diverted to device 4 (printer). Typing 0 also toggles from the screen to the printer. The first value is where the output will be diverted. The second value is the secondary address, and the third is the filename that is used for storing output. It is not necessary to use all parameters.

P (Print Screen)
This command sends the screen display to the printer or disk drive.

Q (Quick Trace)
Same as extramon and micromon.

R (Display Register Contents)
Same as all other monitors listed.

S (Save To Tape Or Disk)
Same as supermon, extramon, and Commodore monitors.

T (Transfer Code)
Same as all other monitors listed.

U (Test Color Ran)
Tests ram for proper function and reports results.

V (Verify Ram Function)
Example: .V 1000 2000
This command tests the section of ram listed above. A '.' will be printed if everything is OK. If a memory location fails the test the address and a binary number showing the data that is not stored properly.

W. (Walk Code)
Same as extramon and micromon.

X (Warm Start Of Basic
X Warm Start Of Basic)
XC (Acts the same as if the computer was turned off then on again.)

(Convert Hexadecimal To Decimal)
Example: .#2048
Answer: .#2049 0801 2049

$ (Convert Hexadecimal To Decimal)
Example: .$0801
Answer: .$0801 2049 0801

+ (Hexadecimal Addition)
Example: .$0801
Example: . + 1000 2000

Answer: . + 1000 2000 =

Adds two hexadecimal numbers together and prints the sum in hex and decimal.

- (Hexadecimal Subtraction)
Example: .- 1000 2000
Answer: .- 1000 2000 — =

Subtracts two decimal numbers and prints the difference in hex and decimal. A positive result may be given when in fact it should be a negative result. The reason is the number does not retain the instruction that the result is negative. To correct the problem type the operands in reverse order. Also the result can be corrected by typing (-0000) and then the positive number that was returned in error. If you enter this information with the return key, the correct result will be displayed.

SUMMARY OF HESMON INSTRUCTION

A (SIMPLE ASSEMBLER)

B (BREAK POINT SET)

C (COMPARE MEMORY)

D (DISASSEMBLE CODE)

E (EXTERNAL RELINKER)

F (FILL MEMORY)

G (GO RUN)

H (HUNT MEMORY)

I (INTERROGATE MEMORY)

L (LOAD FROM TAPE OR DISK)

M (DISPLAY MEMORY IN HEX)

N (NEW LOCATOR)

O (OUTPUT TO DEVICE)

P (PRINT SCREEN)

Q (QUICK TRACE)

R (DISPLAY REGISTER CONTENTS)

S (SAVE TO TAPE OR DISK)

T (TRANSFER CODE)

U (TEST COLOR RAM)

V (VERIFY RAM FUNCTION)

W (WALK CODE)

X (WARM START)

XC (RETURN TO SAME STATE AS WHEN FIRST TURNED ON)

# (CONVERT DECIMAL TO HEXADECIMAL)

$ (CONVERT HEXADECIMAL TO DECIMAL)

+ (ADD TWO HEXADECIMAL NUMBERS)

- (SUBTRACT TWO HEXADECIMAL NUMBERS)

There are three areas of memory that the monitor usually resides. They are basic (0801 hex - 2049 decimal); 8000 hex - 32768 decimal; or C000 hex - 49152 decimal. Most monitors are located in these locations or very close. When using a monitor you must be sure that the program that you load into the monitor is not located in the same area of memory that the monitor resides. This is the reason monitors are located in different areas of memory. You can use a sector editor to determine where in memory your program resides. First use your sector editor to look at track 18. Search each sector of track 18 until you see the name of the program you want. Next count the bytes (a byte is two digits long) that start on far left side of where the name of the program is. Byte 3 contains the track, and byte 4 contains the sector that the program is located on. Example below:

```
Location 00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F

         12 01 82 11 00  4E 41 4D 45  A0 A0 A0 A0  A0 A0 A0
         A0 A0 A0 A0  A0 00 00 00  00 00 00 00  00 00 40 00
```

Byte 3 is the hexadecimal number 11, which is 17 decimal (track 17). Next byte 4 is the hexadecimal number 00 - 00 decimal (sector 0). Now take a look at track 17, sector 0, this is the start of your program. Counting from left to right the second and third bytes contain the address where this program resides in memory. Example below:

```
Location 00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F

         11 04 01 08  A9 00 8D 24  10 A9 15 8D  25 10 00 00
```

Bytes are stored in the low byte, high byte format. This means that the address location is in reverse order. Using this information you can see that our example program resides at 08 01 hex - 2049 decimal (basic). To examine this program you must use a monitor that is not located in basic (0801 hex - 2049 decimal). Most monitors use about 4K of memory.

SUPERMON - (Resides in basic)

EXTRAMON - (There are versions of this monitor located in basic; 50135 decimal; 38083 decimal; and 29900 decimal).

MICROMON - (Usually found to reside at memory location 8000 hex - 49152 decimal).

COMMODORE MONITOR - (There are two versions, one located at 8000 hex - 32768 decimal and C000 hex - 49152 decimal).

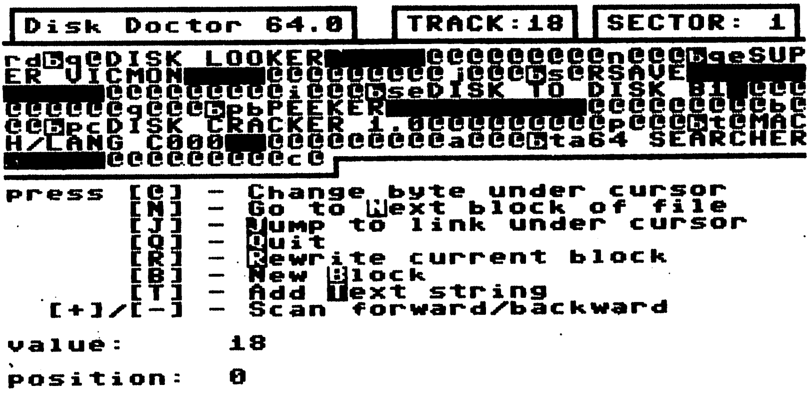
HESMON (cartridge) - (resides at 8000 hex - 32768 decimal)

Monitors located in Basic can be executed with the RUN command or typing SYS2049 (start of basic). All others except Hesmon can be executed by typing SYS 32768 (or what ever decimal location that particular monitor resides.) Hesmon is on cartridge and is located at 8000 hex (decimal 32768.) This monitor is executed by typing SYS32768 or pressing the RESTORE key.

NOTE: There is a sector editor program on the disk that came packed with your disk drive. This is a good program to examine every track and sector on a disk. You can only look, as this program does not have the ability to edit or change data on the disk. A program that can edit the disk can be purchased by MEGASOFT LTD.

DISK DOCTOR INSTRUCTIONS

Disk Doctor 64.1 is a disk manipulator and sector editor. The original Disk Doctor was released into the public domain by SOLIDUS INTERNATIONAL. Since that time it has been modified a little. All versions of the program use the same or slightly modified commands. After loading the program you will see a screen display similar to the one below.



```
Disk Doctor 64.0     TRACK:18     SECTOR:  1

rd▯g@DISK LOOKER        @@@@@@@@@@n@@@▯geSUP
ER VICMON       @@@@@@@@@@j@@@▯s@RSAVE
        @@@@@@@@@@i@@@▯seDISK TO DISK 81 @@@
@@@@@@g@@@▯pbPEEKER              @@@@@@@@@b@
@@▯pcDISK CRACKER 1.0@@@@@@@@@@p@@@▯t@MAC
H/LANG C000  @@@@@@@@@@a@@@▯ta64 SEARCHER
        @@@@@@@@@@c@

press  [@]    -  Change byte under cursor
       [N]    -  Go to Next block of file
       [J]    -  Jump to link under cursor
       [Q]    -  Quit
       [R]    -  Rewrite current block
       [B]    -  New Block
       [T]    -  Add Text string
   [+]/[-]    -  Scan forward/backward

value:       18

position:    0
```

Near the bottom of the screen you will see a prompt for the track and sector. Enter the decimal number of the track and sector you wish to examine. The cursor keys are used to move the cursor to any position within the sector grid.

The purpose of Disk Doctor is to display the contents of any track and sector. The data in a particular sector can be changed, and then rewrote back to the disk or to the same track and sector of another disk. Data may be edited one byte at a time or a string of text can be entered all at once.

[@] - This key is used to change the byte under the cursor to any decimal or ASCII value you wish. Press this key and you will be prompted for a new value. Enter the new value and then press return.

[N] - This key is used to go to the next block of the file. The first two bytes of the sector must have a valid forward pointer, otherwise an error will occur.

[J] - This key is used to jump to the track and sector as indicated by the jump pointer under the cursor. The pointer must point to a valid track and sector otherwise an error will occur.

[Q] - Press this key to exit the program.

[R] - After editing is completed this key is used to rewrite the correct sector back to the disk.

[B] - When this key is pressed you will be prompted to enter a valid track and sector. The requested track and sector will be displayed on the screen.

[T] - Position the cursor over the first byte you wish to edit. Press this key to put the system into text mode. Next enter the desored text, then exit by pressing the return key.

[+ /-]- These keys will allow you to scan forward (+) or backward (-) a sector at a time.

[C]- This key is used to copy a sector to another disk.

CONVERSION CHART

MSB	DECIMAL LSB	HEXADECIMAL	BINARY	ASCII
256	1	01	00000001	
512	2	02	00000010	
768	3	03	00000011	
1024	4	04	00000100	
1280	5	05	00000101	WHITE
1536	6	06	00000110	
1792	7	07	00000111	
2048	8	08	00001000	DISABLES SHIFT/C=
2304	9	09	00001001	ENABLES SHIFT/C=
2560	10	0A	00001010	
2816	11	0B	00001011	
3072	12	0C	00001100	
3328	13	0D	00001101	RETURN
3584	14	0E	00001110	SWITCH/LOWERCASE
3840	15	0F	00001111	
4096	16	10	00010000	
4352	17	11	00010001	CURSOR/DOWN
4608	18	12	00010010	REVERSE/ON
4864	19	13	00010011	CLEAR HOME
5120	20	14	00010100	INSERT/DELETE
5376	21	15	00010101	
5632	22	16	00010110	
5888	23	17	00010111	
6144	24	18	00011000	
6400	25	19	00011001	
6656	26	1A	00011010	
6912	27	1B	00011011	
7168	28	1C	00011100	RED
7424	29	1D	00011101	CURSOR/RIGHT
7680	30	1E	00011110	GREEN
7936	31	1F	00011111	BLUE
8192	32	20	00100000	SPACE
8448	33	21	00100001	!
8704	34	22	00100010	"
8960	35	23	00100011	#
9216	36	24	00100100	$
9472	37	25	00100101	%
9728	38	26	00100110	&
9984	39	27	00100111	'
10240	40	28	00101000	(
10496	41	29	00101001	)
10752	42	2A	00101010	*
11008	43	2B	00101011	+
11264	44	2C	00101100	,
11520	45	2D	00101101	-
11776	46	2E	00101110	.
12032	47	2F	00101111	/
12288	48	30	00110000	0
12544	49	31	00110001	1
12800	50	32	00110010	2
13056	51	33	00110011	3
13312	52	34	00110100	4
13568	53	35	00110101	5
13824	54	36	00110110	6

MSB	DECIMAL LSB	HEXADECIMAL	BINARY	ASCII
14080	55	37	00110111	7
14336	56	38	00111000	8
14592	57	39	00111001	9
14848	58	3A	00111010	:
15104	59	3B	00111011	;
15360	60	3C	00111100	<
15616	61	3D	00111101	=
15872	62	3E	00111110	>
16128	63	3F	00111111	?
16384	64	40	01000000	@
16640	65	41	01000001	A
16896	66	42	01000010	B
17152	67	43	01000011	C
17408	68	44	01000100	D
17664	69	45	01000101	E
17920	70	46	01000110	F
18176	71	47	01000111	G
18432	72	48	01001000	H
18688	73	49	01001001	I
18944	74	4A	01001010	J
19200	75	4B	01001011	K
19456	76	4C	01001100	L
19712	77	4D	01001101	M
19968	78	4E	01001110	N
20224	79	4F	01001111	O
20480	80	50	01010000	P
20736	81	51	01010001	Q
20992	82	52	01010010	R
21248	83	53	01010011	S
21504	84	54	01010100	T
21760	85	55	01010101	U
22016	86	56	01010110	V
22272	87	57	01010111	W
22528	88	58	01011000	X
22784	89	59	01011001	Y
23040	90	5A	01011010	Z
23296	91	5B	01011011	[
23552	92	5C	01011100	
23808	93	5D	01011101	]
24064	94	5E	01011110	^
24320	95	5F	01011111	←
24576	96	60	01100000	
24832	97	61	01100001	
25088	98	62	01100010	
25344	99	63	01100011	
25600	100	64	01100100	
25856	101	65	01100101	
26112	102	66	01100110	
26368	103	67	01100111	
26624	104	68	01101000	
26880	105	69	01101001	
27136	106	6A	01101010	
27392	107	6B	01101011	
27648	108	6C	01101100	
27904	109	6D	01101101	
28160	110	6E	01101110	
28416	111	6F	01101111	
18672	112	70	01110000	
28928	113	71	01110001	
29184	114	72	01110010	
29440	115	73	01110011	
29696	116	74	01110100	

MSB	DECIMAL LSB	HEXADECIMAL	BINARY	ASCII
29952	117	75	01110101	
30208	118	76	01110110	
30464	119	77	01110111	
30720	120	78	01111000	
30976	121	79	01111001	
31232	122	7A	01111010	
31488	123	7B	01111011	
31744	124	7C	01111100	
32000	125	7D	01111101	
32256	126	7E	01111110	
32512	127	7F	01111111	
32768	128	80	10000000	
33024	129	81	10000001	ORANGE
33280	130	82	10000010	
33536	131	83	10000011	
33792	132	84	10000100	
34048	133	85	10000101	F1
34304	134	86	10000110	F3
34560	135	87	10000111	F5
34816	136	88	10001000	F7
35072	137	89	10001001	F2
35328	138	8A	10001010	F4
35584	139	8B	10001011	F6
35840	140	8C	10001100	F8
36096	141	8D	10001101	SHIFT/RETURN
36352	142	8E	10001110	SWITCH/UPPER CASE
36608	143	8F	10001111	
36864	144	90	10010000	BLACK
37120	145	91	10010001	CURSOR/UP
37376	146	92	10010010	REVERSE/OFF
37632	147	93	10010011	CLEAR/HOME
37888	148	94	10010100	INSERT/DELETE
38144	149	95	10010101	BROWN
38400	150	96	10010110	LIGHT RED
38656	151	97	10010111	DARK GREY
38912	152	98	10011000	MEDIUM GREY
39168	153	99	10011001	LIGHT GREEN
39424	154	9A	10011010	LIGHT BLUE
39680	155	9B	10011011	LIGHT GREY
39936	156	9C	10011100	PURPLE
40192	157	9D	10011101	CURSOR/LEFT
40448	158	9E	10011110	YELLOW
40704	159	9F	10011111	CYAN
40960	160	A0	10100000	SPACE
41216	161	A1	10100001	
41472	162	A2	10100010	
41728	163	A3	10100011	
41984	164	A4	10100100	
42240	165	A5	10100101	
42496	166	A6	10100110	
42752	167	A7	10100111	
43008	168	A8	10101000	
43264	169	A9	10101001	
43520	170	AA	10101010	
43776	171	AB	10101011	
44032	172	AC	10101100	
44288	173	AD	10101101	
44544	174	AE	10101110	
44800	175	AF	10101111	
45056	176	B0	10110000	
45312	177	B1	10110001	
45568	178	B2	10110010	

MSB	DECIMAL LSB	HEXADECIMAL	BINARY	ASCII
45824	179	B3	10110011	
46080	180	B4	10110100	
46336	181	B5	10110101	
46592	182	B6	10110110	
46848	183	B7	10110111	
47104	184	B8	10111000	
47360	185	B9	10111001	
47616	186	BA	10111010	
47872	187	BB	10111011	
48128	188	BC	10111100	
48384	189	BD	10111101	
48640	190	BE	10111110	
48896	191	BF	10111111	
49152	192	C0	11000000	
49408	193	C1	11000001	
49664	194	C2	11000010	
49920	195	C3	11000011	
50176	196	C4	11000100	
50432	197	C5	11000101	
50688	198	C6	11000110	
50944	199	C7	11000111	
51200	200	C8	11001000	
51456	201	C9	11001001	
51712	202	CA	11001010	
51968	203	CB	11001011	
52224	204	CC	11001100	
52480	205	CD	11001101	
52736	206	CE	11001110	
52992	207	CF	11001111	
53248	208	D0	11010000	
53504	209	D1	11010001	
53760	210	D2	11010010	
54016	211	D3	11010011	
54272	212	D4	11010100	
54528	213	D5	11010101	
54784	214	D6	11010110	
55040	215	D7	11010111	
55296	216	D8	11011000	
55552	217	D9	11011001	
55808	218	DA	11011010	
56064	219	DB	11011011	
56320	220	DC	11011100	
56576	221	DD	11011101	
56832	222	DE	11011110	
57088	223	DF	11011111	
57344	224	E0	11100000	SPACE
57600	225	E1	11100001	
57856	226	E2	11100010	
58112	227	E3	11100011	
58368	228	E4	11100100	
58624	229	E5	11100101	
58880	230	E6	11100110	
59136	231	E7	11100111	
59392	232	E8	11101000	
59648	233	E9	11101001	
59904	234	EA	11101010	
60160	235	EB	11101011	

MSB	DECIMAL LSB	HEXADECIMAL	BINARY	ASCII
60416	236	EC	11101100	
60672	237	ED	11101101	
60928	238	EE	11101110	
61184	239	EF	11101111	
61440	240	F0	11110000	
61696	241	F1	11110001	
61952	242	F2	11110010	
62208	243	F3	11110011	
62464	244	F4	11110100	
62720	245	F5	11110101	
62976	246	F6	11110110	
63232	247	F7	11110111	
63488	248	F8	11111000	
63744	249	F9	11111001	
64000	250	FA	11111010	
64256	251	FB	11111011	
64512	252	FC	11111100	
64768	253	FD	11111101	
65024	254	FE	11111110	
65280	255	FF	11111111	

Computer memory addresses are comprised of two hexidecimal numbers. Example: (BCBB) the first number is called the MSB (most significant byte), and the second is the LSB (least significant byte.)

HEXADECIMAL TO DECIMAL

To convert hexadecimal addresses to decimal addresses, take the first hexadecimal number (MSB), and using the conversion chart find the decimal equivalent, then multiply the number by 256. Last add this result to the decimal equivalent of the last hexadecimal number (LSB). Example: BC (188 x 256 = 48128) + BB (187) = 48315.

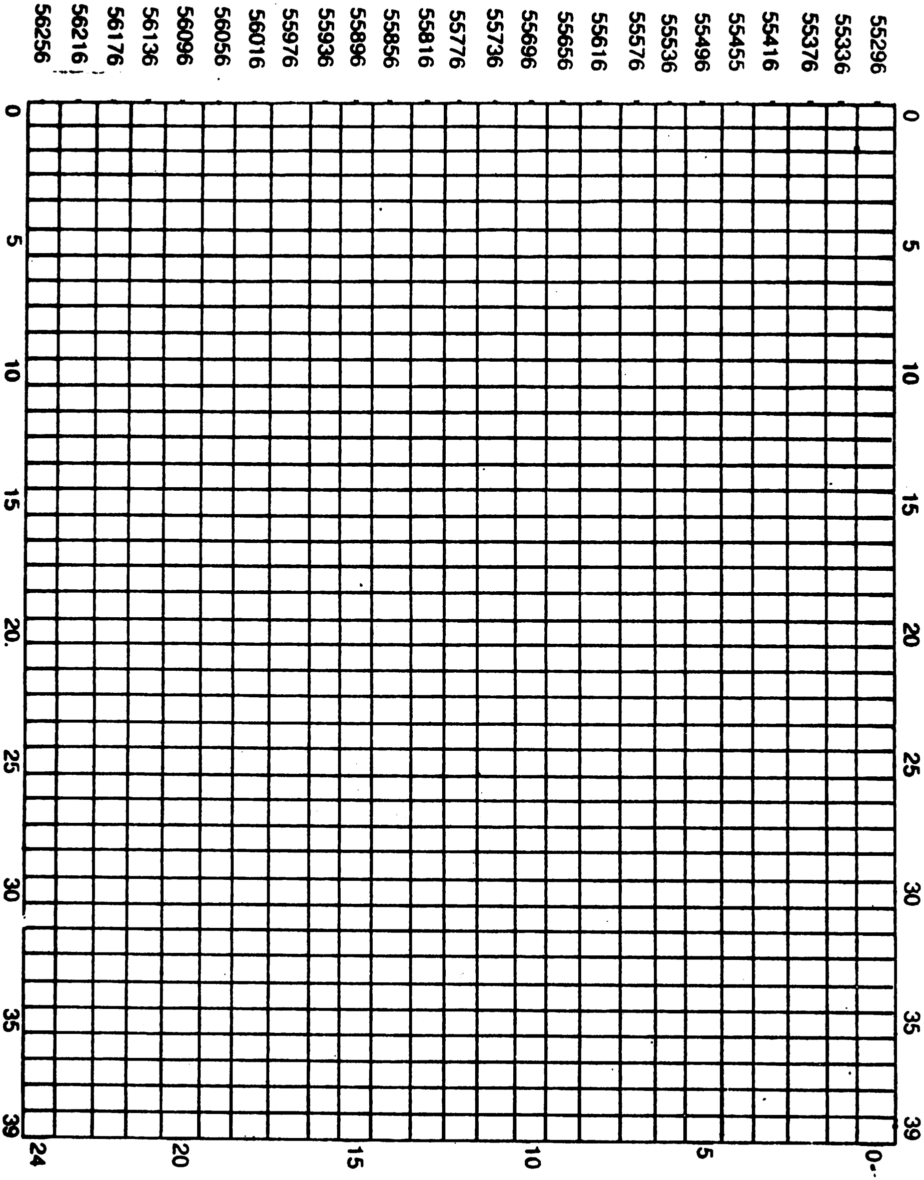
DECIMAL TO HEXADECIMAL

To convert decimal to hexadecimal, look in the MSB column for the decimal number that is less than or equal to the number you are converting. Take this number and subtract it from your number. Then using the result cross reference this number with the hexadecimal equivalent.

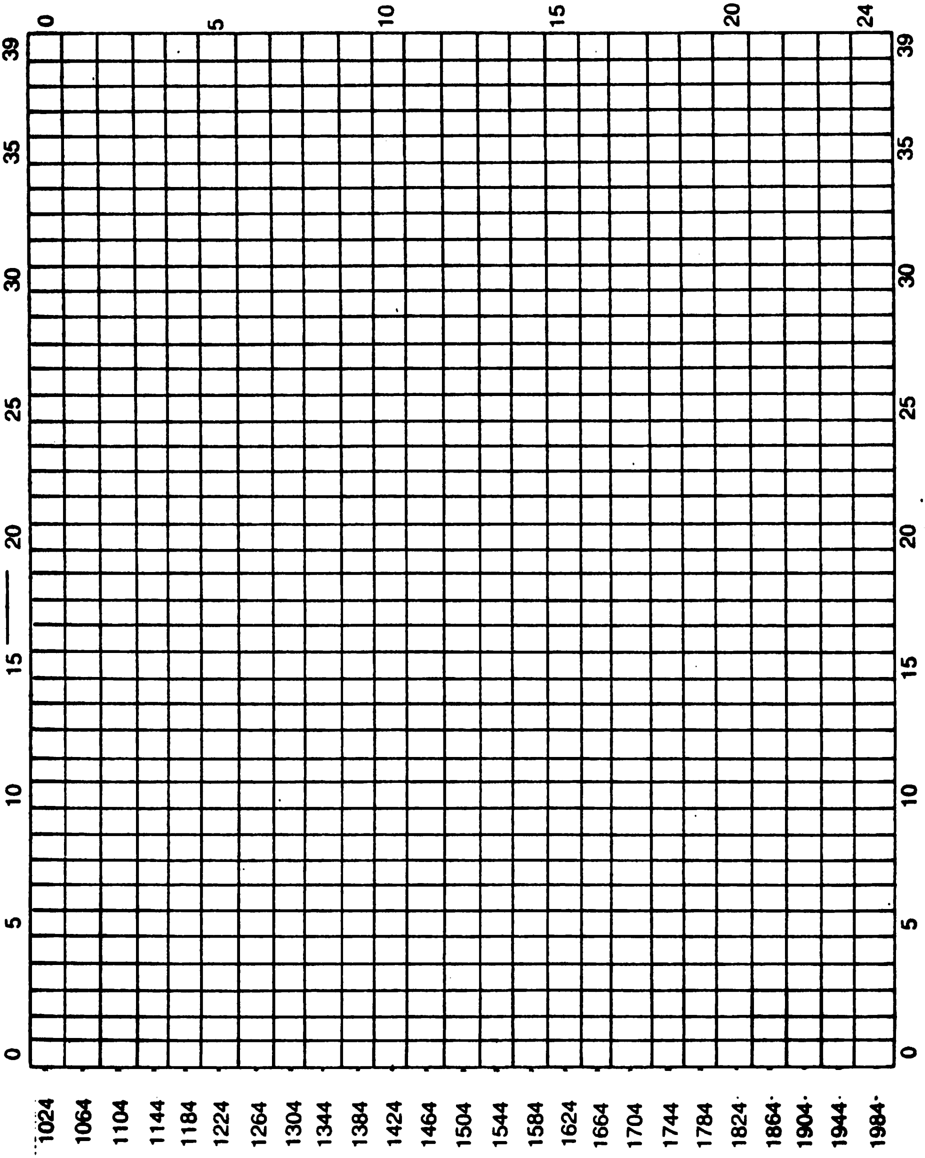
EXAMPLE: Number to convert #32769: 32769 - 32768(80 Hex) = 01(01 Hex)

RESULT: #32769 = $8001

COLOR MEMORY LOCATIONS



SCREEN MEMORY LOCATIONS



TECHNICAL TERMINOLOGY

The most common computer terms and their definitions are in the list below. Please read this list so that you will have a better understanding of your computer and disk drive function.

BAM: (Block Allocation Map). Displays how many blocks have been filled with data and how many are still available for data storage.

BLOCK: The diskette is divided into 683 blocks. Each block contains 256 bytes, and is used to store program data.

BINARY: A binary number contains a combination of zeros and ones that makeup a byte. A byte contains a group of eight bits. The microprocessor of the computer works digitally, distinguishing between two conditions like a switch. The switch being either on or off. Each bit of a byte is either on or off, and the combination of the group contains information that the processor will use to perform a specific function.

BYTE: All letters, number, graphics, and symbols have a numeric equivalent. Every byte contains a two digit hexadecimal number (example 00, 02, A0, BB). See Conversion Chart.

DECIMAL: A numbering system based on ten digits Example (1 2 3 4 5 6 7 8 9 10.)

DIRECTORY: A listing of every file on the diskette (program, relative, sequental or user). Also the location on the diskette where the program starts and how large. The directory is located on track 18 in the 1541 format.

DOS: (Disk Operating System). This controls the internal operation of the disk drive. All functions of data storage between the drive and the computer are performed through this system. Also all operations of diskette formatting.

FILE: A group of blocks that contain data. This data is stored on the diskette in files (program, relative, sequential, and user files).

FORMAT: The method the DOS uses to store information. Before a new diskette can be used, it must be formatted on the system that it is to be used on. All computer systems are not the same, data stored in one format may not be readable by the dos of another system.

HEADER: Contains the disk ID, checksum, sync mark, and all information needed for drive operation. Each sector is made of a header and a block.

MACHINE LANGUAGE: Machine language is two digit hexadecimal code. The most efficient method to program for speed of execution of code. Faster than basic because it does not have to be interpreted by the operating system into code and the processor can understand.

HEX: (Hexidecimal). A number system that is based on sixteen digits. Example (01 02 03 04 05 06 07 08 09 A0 B0 C0 D0 E0 F0 10). Hex numbers are used by the operating system of the computer.

RAM: (Random Access Memory). The part of the computers memory that can be erased, modified, or changed to suit your particular needs. Usually contains a basic or machine language program or instruction.

ROM: (Read Only Memory). A permanent part of the computers memory. Contains the operating system of the computer and cannot be erased or changed. This is the location of memory controls all basic functions.

SECTOR: A track is divided into sectors. The sector will contain the header and the block. All data storage is divided into sectors. The number of sectors per track varies in proportion to the physical size of each track. Outer tracks 1 through 17 contain 21 sectors; tracks 18 through 24 contain 19 sectors; tracks 25 through 30 contain 18 sectors; tracks 31 through 35 contain 17 sectors.

TRACK: A circle on the diskette that is broken up into a number of sectors. There are 35 tracks in the 1541 format. Track one is the outermost and 35 the innermost.

PROTECTION

There are many types of protection that can be used on software. Protection of sofeware may be new to some of you, I will explain a few simple techniques. FIRST* WHAT IS PROTECTION?: Most commercial software uses some form of protection, that inhibit the user from making a copy or modification of their program. The most common forms of protection are listed below, with an explanation of what they do. There are so many types of protection, that it would be impossible to cover them all in just a few pages.

TYPES OF PROGRAM PROTECTIONS:

1. UNLISTABLE PROGRAM LINES
2. INVISIBLE PROGRAM LINES
3. UNLISTABLE PROGRAM
4. DISABLE KEYBOARD
5. COMPILED PROGRAM
6. DONGLES
7. CARTRIDGES
8. AUTO RUN BOOT PROGRAM

TYPES OF DISK PROTECTION:

1. LOOP IN DISK DIRECTOR
2. UNLISTABLE DIRECTORY
3. UNABLE TO FILE COPY
4. INTENTIONAL DISK ERRORS
5. CHANGE BLOCK COUNT OF FILE
6. TRICKY PROGRAM SAVES

PROGRAM PROTECTION

UNABLE TO LIST:

```
10 ?“HI THERE”:REML
20 ?“THIS WILL NOT BE LISTED”:REML
30 ? “NOR THIS”:REML
40 END
```

EXPLANATION: When you try to list this program, you will get a syntax error. If you will notice the rem statement at the end of each line, has a graphic character after it. To produce this character hold down the SHIFT key and press the L key. The basic interpretor cannot understand what the shifted L character does, so it will return a syntax error. It is easy to overcome this problem by listing each line separately or moving the cursor to the line that create the syntax error, and pressing the return key. Now the line will be returned to normal. NOTE-THE QUESTION MARK IS SHORTHAND FOR THE PRINT STATEMENT.

INVISIBLE PROGRAM LINES:

```
10 POKE 775,200:PRINT”
20 ?“THE LINE ABOVE WILL BE HARD TO SEE”
30 ? “ALSO IF THIS PROGRAM IS RUN THE LISTING WILL BE
UNREADABLE”
40 END
```

The program above has two types of protection in line 10. The first is a poke statement that inserts a value of 200 in the list vector, disabling the list function, and second makes line 10 appear invisible. After the poke statement, there is a colon, and then a print statement. After that there is a quotation mark then a space, following is the protection. Put another quotation mark after the space, then press the delete key to erase it. Now hold the SHIFT key down and then the DELETE key (marked INST/DEL) down for a few seconds. You should see the cursor blinking fast, if you do not, retype the line 10 until you do. After you get the blinking cursor, release the SHIFT and DELETE keys. Last press the return key 25 times, you should see a reversed. Now enter line 10 by pressing the return key. When the program is run you will not be able to get a listing, and line 10 will be invisible. Actually line 10 is listed, and then deleted. If you look close, you will see it blink on the screen for a very short time. To overcome this type of protection, line 10 can be deleted, providing it has no information that is required for the program to operate. To make line 10 visable, the line has to be retyped, leaving out the deletes. NOTE-THIS TYPE OF PROTECTION CAN BE ANYWHERE IN YOUR PROGRAM.

UNLISTABLE PROGRAM:

```
10 POKE808,225
20 ?“ANYTHING”
30 GOTO 20
```

EXPLANATION: After the program above has been run, and not before, you will be unable to stop it. This is because, the poke in line 10, disables the RUN/STOP and RESTORE keys. Also when you attempt to get a program listing it will be unreadable.

DISABLE KEYBOARD:

```
10 POKE649,0
20 ?"ANYTHING"
30 GOTO 20
```

EXPLANATION: Line 10 pokes memory location 649 with a value of zero. 649 is the location of the keyboard buffer. Putting a zero here will keep the buffer from holding any keyboard entry.

COMPILED PROGRAMS:

To compile a program, first it has to be written in basic. When a program has been compiled, it is rewrote in what is called p code. This type of code is similar to machine language, but not identical. A compiler program is required to compile a basic program. There are a number of programs on the market that will compile basic. If you wish to purchase one of these programs call MEGASOFT LTD. Protection is not the only reason for compiling a basic program. A compiled program will execute much faster than its basic counter part. A compiled program cannot be listed like a basic program can. You must use a memory monitor to examine the code and even then it is very difficult if not impossible to follow. This makes it a very good place to hide your program protection. The only way to get around this type of protection is to find a decompiler program, if one exists.

DONGLES:

A dongle is a device that is plugged into the joystick port. This is how it works. When the program is developed, it has a routine to check for the presents of the dongle if it is not present the program will crash. This type of protection is very difficult to overcome. There are two routes you can go, the first is to duplicate the dongle. This can be very difficult and require special equipment. The other way is to look at the programs code and try to find the section that contains the protection which can be quite a challenge. Use the Commodore reference guide to find out what each pin in the port is used for (page 395). Also in the reference guide see page 343 and pages 328 - 329 for the memory location that controls the joystick ports. This may help in finding the code that is used to check for the dongle. If the program does something with memory locations DC01 or D419, you have probably found the right area.

CARTRIDGES:

When the computer is turned on it checks to see if locations 32768 (hex 8000) through 32777 (hex 8009), contain the code CBM80. If this code is present it checks the game line (#8), if a cartridge is present the computer will allocate memory from $8000 to $9FFF (8K). If (16K) then the value in memory location 0001 is changed from 37 to 36, this flips out basic and all memory from $8000 through BFFF is allocated.

It is possible to copy what is on the cartridge to disk, through the use of a memory monitor. First you must have a method to insert the cartridge with out the program executing. A good way to do this is by using a mother board. A mother board is a hardware device that has a number of cartridge slots, each can be activated by a switch. Also most mother boards have a reset switch, this is a must. Turn off the computer, insert the mother board, plug in the cartridge, make sure that the switch is off. Turn on the computer, flip on the switch to activate the cartridge. Now turn off the cartridge switch. Press the reset switch to regain control of the computer. The cartridge program will still be in memory. Now load a memory monitor that is located at C000. Execute the monitor by typing SYS49152. Look at memory from $8000 to BFFF, it may be necessary to switch off basic to see the code. Type M 0001, change the 37 to 36. Examine the code now and look for any code that is trying to change the program at $8000 or above this is most likely the protection. After you have found this code, change any bytes that are protection to the hex value EA (nop) which means no operation. Next insert a preformatted diskette into the drive, and type S ''name''08,800,C000 (for 16K), then press return. This part of memory will be saved to disk. Try to load and run the program if it does not work it is possible that there is protection you may not have changed. Repeat the process over, until you are successful. NOTE - It is possible to execute the program from within the monitor. First change the byte at $8007 from 38 to 30, this will enable you to regain control after the program is executed. Locations $8000 to $8001 contain the cold start vectors, and $8002 to $8003 are the warm start vectors. First execute the cold start vectors, it may be necessary to reload the program and use the warm start vectors. Also before closing, it would be a good idea to transfer the code from $8000 through BFFF (16K) to $2000. Then using the cold start vectors first, execute the program at $8000. After the program crashes, reset the computer, then type SYS49152 to re-enter the monitor. Use the compare command from the monitor to find any memory locations that were changed from the monitor to find any memory locations that were changed when the program was executed. Any bytes in memory that were changed is a good place to start looking. ROM (read only memory) cannot be changed, but when the code is transfered to disk it is no longer in ROM, and can be changed. The code is now in RAM (random access memory) and code that would change any bytes in memory from $8000 to $BFFF is most likely protection.

AUTO RUN PROGRAM:

Another good form of protection is by loading your main program through an auto run program. An auto run program when loaded into memory, will start to execute the code without typing run. This is a good form of protection because your program will be loaded into memory, and then run. If you disable the RUN/STOP and RESTORE function within your program, then it will not be very easy to make any changes. There is a program available through MEGASOFT LTD that will make an auto run boot program for you. If you wish to write your own, the November 1984 COMPUTE GAZETTE has a disk auto load program you can type in.

DISK PROTECTION

LOOPING DIRECTORY:

The directory of a disk is on track 18, all disk drive functions start with track 18. You will need a sector editor to view the data on a disk, sector editors can be found in the public domain, free of charge. The sector editor is a program that is designed to view and modify the track and sectors of a disk. Disk doctor is a sector editor that is the public domain, and there are many more commercial sector editors on the market. I would suggest that you acquire this type of software, if you plan to do much work with the disk drive. Assuming you have a sector editor, load and run it. Insert a disk that has programs on it, please be sure that you have a backup of this disk, in the event that you make a mistake, and destroy it. You should see a prompt requesting which track and sector you want to view. Enter track 18, sector 1, you will see a display similar to the one on page 37. The first two bytes 12 and 04, are the pointers to the next block and sector. The 12 is the hexidecimal equivalent of the decimal number 18, which is track 18, and 04 is sector 04. These first two bytes are the pointers to the next sector that has files stored on it. If this was the last sector of the directory file the two bytes would be the hexidecimal OO FF, meaning that this is the end of the directory files. The number of sectors that are used for the directory depends on how many programs are on the disk. To put a loop in the directory we will change the first two bytes on the last sector of the directory file from OO FF to 12 01 (hexidecimal for 18 01). Having done this the disk drive will search in vain for the last sector on the directory file. This will not prevent programs on the disk from loading and running, only from listing the directory.

DIRECTORY FIX:

To fix a directory that has a loop, find where the first two bytes point to track 18 sector 01. Then change them to the hexidecimal numbers 00. FF. Now the directory is back to normal.

UNLISTABLE DIRECTORY

Load and run your sector editor, go to track 18, sector 00. Each byte is two numbers long (example 12), now count the bytes on track 18 sector 00. Bytes 144 through 161 contain the name of the disk, change the first six bytes to 14, 14, 14, 00, 00, 00. Now the directory is unlistable, change the six bytes back to what they were before to have a directory that lists.

UNABLE TO FILE COPY:

You can stop most file copy programs from copying your files very simply. Change the name of the disk along with the ID and 2A to unprintable characters. Add special characters after the first A0 in the file name. Put fake programs on the diskette that will put the drive in an endless loop. See page 34 for an example of these techniques.

DISK ERROR PROTECTION

One of the most used forms of protection consists of putting intentional errors on the disk. A list and description of these errors is below.

1. **20** (Read error, the disk drive is unable to locate the data block header.)
2. **21** (Read error, the disk drive is unable to find a sync mark on the track.)
3. **22** (read error, the disk drive has been told to read or verify a data block that was not written correctly.)
4. **23** (Read error, the disk drive has found a checksum error in a data block.)
5. **27** (read error, the disk drive has found a checksum error in header.)
6. **29** (Disk ID MISMATCH)

It works like this, first the program on the disk has a routine that checks for a particular error condition on the disk. Then the read/write head of the disk drive is sent out to a specified track and sector to check if this error is present. If the error is not present the program is directed to crash. The protection routine below can be used in your software.

```
10010 T=1:S=1
10020 OPEN 15, 8, 15, “I”:REM OPEN ERROR CHANNEL AND INITIALIZE DRIVE
10030 OPEN 8,8,8:REM OPEN CHENNEL FOR DATA
10040 PRINT#15,“U1:8,0”;T;S:REM BLOCK READ, USING CHANNEL 8,DRIVE ?
10050 INPUT#15,A$,B$,C$,D$:REM REAR ERROR CHANNEL
10060 CLOSE8: CLOSE15: REM CLOSE ALL FILES
10070 IF VAL(A$) < > 23 THEN 64760:REM IF ERROR VALUE NOT PRESENT THEN RESET
10080 RETURN:REM RETURN TO START OF PROGRAM
```

Amend this subroutine to your program. Then use a gosub statement to check for an error on the disk. Next you have to create an error 23 on the disk using a routine out of the book INSIDE THE COMMODORE DOS by RICHARD IMMERS AND GERALD G. NEUFELD. If this is not available MEGASOFT LTD. markets a program that will put any type of error on your disk.

CHANGE BLOCK COUNT OF FILE:

This technique can be used to hide the size of your program. This will confuse someone who copys your files. To physically change the block county modify bytes 30 and 31 of the file you wish to hide. This will make it difficult to determine the actual size of the program. See explanation of this on page 35.

TRICKY PROGRAM SAVES:

Program files can be saved to disk as sequential or user files. This type of protection will confuse someone looking at your files. The method used to do this is explained below.

Save"0:name,s"8 (save as a seq file)

To load this program type in the following - Load "0:name,s",8

Substitute a (U) in place of the (S) if you wish to change to user files.

EXPLANATION OF TRACK 18, SECTOR 0:

The example on the left side of page 45 is a normal track 18, sector/block 0. The display on the right contains a form of disk protection. Track 18, sector 0 is where the BAM is located. There are 256 bytes per sector, starting with zero and ending in 255. All numbers are displayed in hexadecimal.

Bytes 0 and 1 are the pointers to the next track and sector. Byte 0 (12 hex or 18 decimal) is the track and byte 1 (01 hex or 01 is the sector.

The second byte is the ASCII character A (41 hex). This byte indicates that this disk has the 1541 format. If this byte is changed to any other value the disk will be write protected. The drive will read the disk but will not be able to write to it. See the modified example on page 45. Byte 2 has been changed from an A (41 hex) to a graphic character (65 hex). This will stop the drive from writing to this disk. However, it is still possible to reformat the disk.

The third byte is unused.

Bytes 4 through 143 are the BAM. There are four bytes per track, the first byte in each track contains the number of blocks free in that track. The three bytes that follow contain a bit map of the available sectors. The bit map refers to a series of 1's and 0's. Each 1 meaning that the bit is on and a 0 is off. In the example below the first byte is 15 hex (21 decimal), this tells users that there are 21 blocks free. If you count the bit map below you will see that there are 21 bits on. Changing the first byte to another value will give a false number of blocks free. A zero in this location will tell the drive that there are no blocks free. You can also insert any other value such as FF, and the drive will be fooled into thinking that there are 255 blocks free. However it will not be possible to store data in these imaginary blocks. By changing the values of the last three bytes in each track, the drive will be unable to determine which blocks are available. Validating the disk will restore these byte to normal. Changing the BAM will not affect the operation of your programs. NOTE - Use the conversion chart in this book to find the binary equivalent of a hexadecimal value. This will be used to determine the bit map of a track.

```
EXAMPLE: HEX 15      F    F      F    F      1    F
BIT MAP OF TRACK: 1111 1111  1111 1111  0001 1111
```

Bytes 144 through 159 are used for the disk name. If the name is less than sixteen characters the balance is filled with shifted spaces. Bytes 160 and 161 are shifted spaces. The disk ID are bytes 162 and 163, and byte 164 is a shifted space. Bytes 165 and 166 are the DOS version and format type. The bytes 167 to 170 are shifted spaces, and bytes 170 through 255 are unused. In the modified example of track 18, sector 0, bytes 144 through 149 have been changed to (14 14 14 00 00 00.) This modification is used to make the directory unlistable. Also I have used graphic characters for the disk ID and format type. This will create problems when you attempt to list the directory. There are a number of modifications you can do to bytes 144 through 159 that will cause the directory listing to react strangely. Try putting different colors values in this location, or clear the screen using the hex equivalent of clear/home (93). Use the conversion chart in this book as a reference guide when modifying these bytes. Experiment using different values i these locations, and watch the results. It is recommended that you use a disk that is expendable.

EXPLANATION OF TRACK 18, SECTOR 1:

Track 18, sector 1 is where the directory begins. Bytes 0 and 1 contain the pointers to the track and sector of the next block of the directory file. Byte 2 is the hex value for the type of file. Below is a list of the different types of files.

Hex Value	**File Type**
$00	Scratched
$80	Deleted
$81	Sequential
$82	Program
$83	User
$84	Relative

AN * before the file type indicateds that it is an unclosed file.

An after the file name indicates that the file is locked. NOTE - This type of file cannot be scratched.

Bytes 3 and 4 are the pointers to the start of the file.

Byte 3 is the track and 4 is the sector.

Bytes 5 through 20 contain the file name (can be up to sixteen characters), if there are less than sixteen characters the balance will be filled with shifted spaces.

Bytes 21 to 23 are used for relative files. Bytes 24 through 27 are unused. Bytes 28 and 29 are used to store the track and sector of the replacement file, when the save and replace command is used.

Bytes 30 and 31 contain the number of blocks that are used by the file. Value calculated in the low byte, high byte. EXAMPLE: Assume the bytes are (FF 00), FF equals 255 decimal, and 00 equals 0 decimal, (255 is the size of the file). If you change the value of these bytes you can hide the actual size of your program. Modifying these bytes will not affect the programs ability to load.

The remaining files have the same format. The first two bytes of the last block of the directory file are 00 FF (indicates to the drive that this is the last block in the directory. If the values 12 01 are put in this block, the drive will be told to go to track 12 hex (18 decimal), sector 01 hex (01 decimal). This will put the drive in an endless loop. NOTE - If you see this type of protection, you can defeat it by restoring the end of the file marker (00 FF).

Another type of protection is to insert special characters following the file name and after the first shifted space (AQ).

EXAMPLE:

```
12 04 82 11   00 48 49 20   54 48 45 52   45 A0 93 1F
A0 A0 A0 A0   A0 00 00 00   00 00 00 00   00 00 50 00

ASCII

.  .  .  .    .  .  H  I       T  H  E  R     E  .  .  .
                 .  .  .    .  .  .  .     .  .  P  .
```

In the example above I have inserted a clear/home and then the value for the color blue. This will foul up most file copy programs. Try these values and others and watch the results, it should prove to be interesting.

NOTE - ON PAGES 37 and 38 YOU WILL FIND TYPICAL EXAMPLES OF TRACK 18, SECTOR 1 AND TRACK 18, SECTOR 4.

ORIGINAL

```
TB/EDITOR  DISK=8  TRK=18 BLK=0  MODE=H

  0 1 2 3  4 5 6 7  8 9 A B  C D E F

0 12014100 15FFFF1F 15FFFF1F 15FFFF1F
1 15FFFF1F 15FFFF1F 15FFFF1F 15FFFF1F
2 15FFFF1F 15FFFF1F 15FFFF1F 15FFFF1F
3 15FFFF1F 15FFFF1F 11D75F1F 00000000

4 00000000 00000000 10ECFF07 00000000
5 00000000 12BFFF07 13FFFF07 13FFFF07
6 13FFFF07 12FFFF03 12FFFF03 12FFFF03
7 12FFFF03 12FFFF03 12FFFF03 11FFFF01

8 11FFFF01 11FFFF01 11FFFF01 11FFFF01
9 31353431 54455354 2F44454D 4FA0A0A0
A A0A05A58 A03241A0 A0A0A000 00000000
B 00000000 00000000 00000000 00000000

C 00000000 00000000 00000000 00000000
D 00000000 00000000 00000000 00000000
E 00000000 00000000 00000000 00000000
F 00000000 00000000 00000000 00000000
```

```
TB/EDITOR  DISK=8  TRK=18 BLK=0  MODE=A

  0 1 2 3   4 5 6 7   8 9 A B   C D E F

0 . . A .   . π π .   . π π .   . π π .
1 . π π .   . π π .   . π π .   . π π .
2 . π π .   . π π .   . π π .   . π π .
3 . π π .   . π π .   . O . .   . . . .

4 . . . .   . . . .   . ▪ π .   . . . .
5 . . . .   . ▚ π .   . π π .   . π π .
6 . π π .   . π π .   . π π .   . π π .
7 . π π .   . π π .   . π π .   . π π .

8 . π π .   . π π .   . π π .   . π π .
9 1 5 4 1   T E S T   / D E M   O
A     Z X     2 A           .   . . . .
B . . . .   . . . .   . . . .   . . . .

C . . . .   . . . .   . . . .   . . . .
D . . . .   . . . .   . . . .   . . . .
E . . . .   . . . .   . . . .   . . . .
F . . . .   . . . .   . . . .   . . . .
```

MODIFIED

```
TB/EDITOR  DISK=8  TRK=18 BLK=0  MODE=H

  0 1 2 3  4 5 6 7  8 9 A B  C D E F

0 12016500 15FFFF1F 15FFFF1F 15FFFF1F
1 15FFFF1F 15FFFF1F 15FFFF1F 15FFFF1F
2 15FFFF1F 15FFFF1F 15FFFF1F 15FFFF1F
3 15FFFF1F 15FFFF1F 11D75F1F 00000000

4 00000000 00000000 10ECFF07 00000000
5 00000000 12BFFF07 13FFFF07 13FFFF07
6 13FFFF07 12FFFF03 12FFFF03 12FFFF03
7 12FFFF03 12FFFF03 12FFFF03 11FFFF01

8 11FFFF01 11FFFF01 11FFFF01 11FFFF01
9 14141400 0000A0A0 A0A0A0A0 A0A0A0A0
A A0A06565 A06565A0 A0A0A000 00000000
B 00000000 00000000 00000000 00000000

C 00000000 00000000 00000000 00000000
D 00000000 00000000 00000000 00000000
E 00000000 00000000 00000000 00000000
F 00000000 00000000 00000000 00000000
```

```
TB/EDITOR  DISK=8  TRK=18 BLK=0  MODE=A

  0 1 2 3   4 5 6 7   8 9 A B   C D E F

0 . . ‾ .   . π π .   . π π .   . π π .
1 . π π .   . π π .   . π π .   . π π .
2 . π π .   . π π .   . π π .   . π π .
3 . π π .   . π π .   . O . .   . . . .

4 . . . .   . . . .   . ▪ π .   . . . .
5 . . . .   . ▚ π .   . π π .   . π π .
6 . π π .   . π π .   . π π .   . π π .
7 . π π .   . π π .   . π π .   . π π .

8 . π π .   . π π .   . π π .   . π π .
9 . . . .   . .
A     ‾ ‾     ‾ ‾           .   . . . .
B . . . .   . . . .   . . . .   . . . .

C . . . .   . . . .   . . . .   . . . .
D . . . .   . . . .   . . . .   . . . .
E . . . .   . . . .   . . . .   . . . .
F . . . .   . . . .   . . . .   . . . .
```

```
TB/EDITOR  DISK=8  TRK=18 BLK=1  MODE=H

  0 1 2 3  4 5 6 7  8 9 A B  C D E F

0 12048211 00484F57 20544F20 555345A0
1 A0A0A0A0 A0000000 00000000 00000D00
2 00008211 03484F57 20504152 54205457
3 4FA0A0A0 A0000000 00000000 00000500

4 00008211 09564943 2D323020 57454447
5 45A0A0A0 A0000000 00000000 00000400
6 00008213 00432D36 34205745 444745A0
7 A0A0A0A0 A0000000 00000000 00000100

8 00008213 01444F53 20352E31 A0A0A0A0
9 A0A0A0A0 A0000000 00000000 00000400
A 00008213 03434F50 592F414C 4CA0A0A0
B A0A0A0A0 A0000000 00000000 00000B00

C 00008213 09505249 4E544552 20544553
D 54A0A0A0 A0000000 00000000 00000900
E 00008210 00444953 4B204144 44522043
F 48414E47 45000000 00000000 00000400
```

```
TB/EDITOR  DISK=8  TRK=18 BLK=1  MODE=A

  0 1 2 3  4 5 6 7  8 9 A B  C D E F

0 . . . .  . H O W    T O    U S E
1            . . .  . . . .  . . . .
2 . . . .  . H O W    P A R  T   T W
3 O          . . .  . . . .  . . . .

4 . . . .  . V I C  - 2 0    W E D G
5 E          . . .  . . . .  . . . .
6 . . . .  . C - 6  4   W E  D G E
7            . . .  . . . .  . . . .

8 . . . .  . D O S    5 . 1
9            . . .  . . . .  . . . .
A . . . .  . C O P  Y / A L  L
B            . . .  . . . .  . . . .

C . . . .  . P R I  N T E R    T E S
D T          . . .  . . . .  . . . .
E . . . .  . D I S  K   A D  D R   C
F H A N G  E . . .  . . . .  . . . .
```

```
TB/EDITOR  DISK=8  TRK=18 BLK=4  MODE=H

   0 1 2 3  4 5 6 7  8 9 A B  C D E F

 0 00FF8210 01444952 A0A0A0A0 A0A0A0A0
 1 A0A0A0A0 A0000000 00000000 00000400
 2 00008210 03564945 57204241 4DA0A0A0
 3 A0A0A0A0 A0000000 00000000 00000600

 4 00008210 07434845 434B2044 49534BA0
 5 A0A0A0A0 A0000000 00000000 00000400
 6 00008210 0F444953 504C4159 20542653
 7 A0A0A0A0 A0000000 00000000 00000E00

 8 00008214 02504552 464F524D 414E4345
 9 20544553 54000000 00000000 00000900
 A 00008214 07534551 55454E54 49414C20
 B 46494C45 A0000000 00000000 00000500

 C 0000820F 0152414E 444F4D20 46494C45
 D A0A0A0A0 A0000000 00000000 00000D00
 E 00000000 00000000 00000000 00000000
 F 00000000 00000000 00000000 00000000
```

```
TB/EDITOR  DISK=8  TRK=18 BLK=4  MODE=A

   0 1 2 3  4 5 6 7  8 9 A B  C D E F

 0 . π . .  . D I R
 1            . . .  . . . .  . . . .
 2 . . . .  . V I E  W   B A  M
 3            . . .  . . . .  . . . .

 4 . . . .  . C H E  C K   D  I S K
 5            . . .  . . . .  . . . .
 6 . . . .  . D I S  P L A Y    T & S
 7            . . .  . . . .  . . . .

 8 . . . .  . P E R  F O R M  A N C E
 9   T E S  T . . .  . . . .  . . . .
 A . . . .  . S E Q  U E N T  I A L
 B F I L E    . . .  . . . .  . . . .

 C . . . .  . R A N  D O M    F I L E
 D            . . .  . . . .  . . . .
 E . . . .  . . . .  . . . .  . . . .
 F . . . .  . . . .  . . . .  . . . .
```

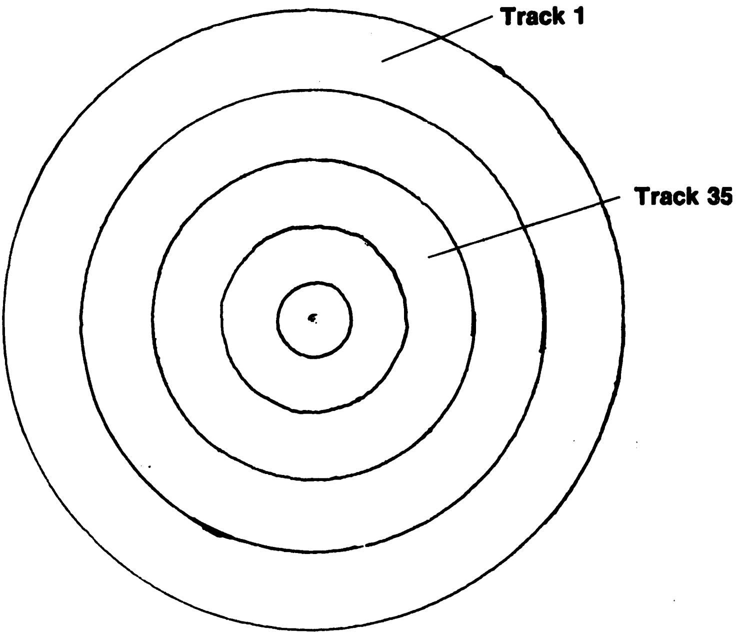
DISK EXAMINATION

To enable you to understand the diskette, I have provided you with the layout of a track, sector, header block, and data block. On page 40 you will find this information. The Commodores DOS formats a diskette with 35 tracks. Each track is divided into sectors/blocks. As you will se on page 40 that every track does not have the same number of sectors. There are four different size tracks. Track 1 is located on the outside edge of the diskette and each successive track follows working toward the inside. Since the outside tracks are larger than the inside tracks the amount of sectors per track varies. Since the speed of the drive is the same for all tracks (approximately 300 rpm). The smaller tracks are read at a different speed. When the diskette is formatted the DOS encodes each sector so it can follow where data is to be placed on the diskette (see page 40). Each sector is comprised of two parts.

1. **Header Block [identifies sector]**
2. **Data Block [holds data]**

Some types of disk protection modify the format of a sectors layout. Then create a special DOS routine to read it. This type of protection is best left to the experienced programer.

DISK LAYOUT (not to scale)



LAYOUT OF TRACKS:

Track Number	Number of Sectors
1 to 17	0 to 20 (21)
18 to 24	0 to 18 (19)
25 to 30	0 to 17 (18)
31 to 35	0 to 16 (17)

The diskette has a total of 683 blocks, of which only 664 can be used for data storage. This is because track 18 has 19 sectors/blocks that can only be used to store the directory and location of files.

LAYOUT OF SECTORS:

Example: Sector #0

::sync mark:: header block::data block::inter sector gap

::sync mark::

HEADER BLOCK LAYOUT:

No ot Bytes	
0	sync mark
1	header block ID
1	header block checksum
1	sector number
1	track number
1	ID character 2
1	ID character 1
2	off bytes used for padding when formatting header gap

DATA BLOCK LAYOUT:

No. of Bytes	
0	sync character
1	data block ID
256	data bytes
1	data block checksum
2	off bytes used for padding before writing
varies	inter sector gap

MENU PROGRAM

The purpose of a menu program is used to load any program on the diskette. This type of program is more convenient than listing the directory and then loading the program you wish. Below you will find a listing for a menu program. An explanation will follow the listing.

```
100 POKE53280,0:POKE53281,0:POKE646,6:PRINTCHR$(147)
120 CLR
125 PRINT"▩▩▩▩≡READING DISK DIRECTORY:▣"
130 PRINT"________________________________________"
140 DIMA$(100)
170 OPEN5,8,0,"$"
190 GET#5,B$:IFST<>0THEN270
200 IFB$<>CHR$(34)THEN190
210 A$=""
220 GET#5,B$:IFB$<>CHR$(34)THENA$=A$+B$:GOTO220
230 GET#5,B$:IFB$=CHR$(32)THEN230
240 A$(X)=A$:PRINTX;"▔"
250 GET#5,B$:IFB$<>""THEN250
260 IFST=0THENX=X+1:GOTO190
270 CLOSE5
290 N=10
310 PRINT"□▪▪▪▪▪▪▌≡D I S K>>>    ";"▣"A$(0)"▣▩"
315 GOSUB 1000
320 IFN<10THENN=10
330 IFN>100THENN=100
335 PRINT"▣▩▩"
340 FORP=(N-9)TON:PRINT"▪▪▪▪▪▪▪▪▪▪▪▪▌"P;A$(P):NEXTP
350 PRINT"▪▪▪▪▪▪▪▩▩PRESS ≡N▣   FOR NEXT SCREEN▩"
360 PRINT"▪▪▪▪▪▪▪PRESS ≡L▣   FOR LAST SCREEN▩▩▩▩"
370 PRINT"▣PRESS RETURN THEN MAKE YOUR SELECTION"
400 GETZ$:IFZ$=""THEN400
410 IFZ$="N"THENN=N+10:GOTO310
420 IFZ$="L"THENN=N-10:GOTO310
480 INPUT"≡▩ENTER NUMBER  ▪▪▪▪▌";S
500 IFS<1ORS>XTHEN480
510 IFLEN(A$(S))>12THENA$(S)=LEFT$(A$(S),12)+"*"
520 PRINT"LOAD"+CHR$(34)+A$(S)+CHR$(34)+",8,1▔▔▔"
550 POKE631,13:POKE632,82:POKE633,85:POKE634,78:POKE635,13:POKE198,5
560 END
1000 PRINT"          ╭──────────────────────╮"
1010 FORA=1TO10:PRINT"          |                      |"
1020 NEXT
1030 PRINT"          ╰──────────────────────╯"
1040 RETURN
```

EXPLANATION: You can save this program to your disk using a file name of menu. When the program is loaded and run, it will read the directory on the disk and display the files on the screen ten at a time. It is not required that you put this program on every disk. Only that you insert the disk you wish load a file off into the drive before running the menu program.

This program can be modified as you wish. You may wish to change the colors or the size of the directory displayed on the screen. I will not explain the program line by line only the important sections. First the program sets the screen and cursor colors (change to any color you like.) the screen is cleared and lines 30 and 40 are displayed. Next the variable A$ is DIMensioned to reserve space at the end of basic memory. A data channel is opened to read the directory as a sequential file. Then the get# statement is used to examine the directory. Lines 70 through 150 read the directory. Line 160 sets up the amount of files to be displayed. The next line displays the disk name, then goes to line 1000. Line 1000 creates a border around the ten program files that are displayed.

Type line 1000 in as follows: print "type 8 spaces; hold shift key down and press the U key; hold the shift key down and press the C key; hold the shift key down and press the I key; ". Type line 1010 in as follows: for a = 1 to 10:print"type 8 spaces; hold shift key down and press the B key; type 21 spaces; hold the shift key down and press the B key;". Line 1010 is printed ten times. Line 1030 is the same as line 1000 except for press the shifted J key instead of the U key. Also substitute a shifted K key for the I key.

Line 1040 returns the program to line 190. Lines 190 to 250 sets up display of files and screen layout. Lines 260 to 290 scans keyboard for either the N or L keys to be pressed. These keys control which set of ten file will be displayed (if less than ten files are on the disk then only numbers will be displayed. Line 300 waits for file number input. Line 310 to 330 print (load "filename",8,1 to the screen). Line 340 inserts a return, the word RUN and another return into the keyboard buffer. Then the poke198,5 executes the keyboard input. This will run the program after it is loaded into memory.